An Iterated Local Search to find many solutions of the 6-states Firing Squad Synchronization Problem

Manuel Clergue, Sébastien Verel, Enrico Formenti

An Iterated Local Search to find many solutions of the 6-states Firing Squad Synchronization Problem

Manuel Clergue*,[a], Sébastien Verel[b], Enrico Formenti[c]

[a] *Université des Antilles, EA 4540 - LAMIA - Laboratoire de Mathématiques Informatique et Applications, Pointe à Pitre, France*
[b] *Univ. Littoral Côte d'Opale, EA 4491 - LISIC - Laboratoire d'Informatique Signal et Image de la Côte d'Opale, F- 62228 Calais, France*
[c] *I3S, Université Nice Sophia Antipolis, UMR CNRS 7271, 2000 route des Lucioles, BP 121, 06903 Sophia Antipolis cedex, France*

Abstract

This paper proposes an optimization approach for solving a classical problem in cellular automata theory: the 6-states Firing Squad Synchronization Problem (FSSP). To this purpose, we introduce an original optimization function which quantifies the quality of solutions according only to the main goal of the problem without taking into account any side information about cellular automata computations. This function is used for a dedicated Iterated Local Search algorithm which finds several hundreds of new solutions of the FSSP. Note, that up to present only one human-designed solution was known which is optimal in time. Most of the new solutions found by our algorithm have lower complexity (in terms of number of transitions rules used). An analysis of the fitness landscape for FSSP explains why, counter-intuitively, local search strategy can achieve good results for FSSP.

Key words: Firing Squad Synchronization Problem, Iterated Local Search, Metaheuristics, Fitness landscape, Cellular automata, Complex Systems

1. Introduction

When facing a computational problem, one can more or less easily describe the desired outputs or the final state of the system. However, it is often much more difficult to design the process that allows to obtain those outputs or to reach a final state. That calls for the need of computer scientist or IT engineers. Automatic generation of programs, and one of its most representative instances, genetic programming, is a scientific field that aims to develop methods for goal-driven design of systems [1]. Those methods should be able to generate systems just from the specification of the desired final states and from the specification of its environment, including rules, constraints, and so on.

One possibility consists of considering the design as an optimization problem, in which one try to maximize how much, or how well, the system meets its requirements. An optimal solution of this optimization problem provides a solution of the computational problem. Early successes in genetic programming [2], for example, should be considered as an encouragements to follow this kind of methodology.

However there are still some issues that have to be addressed in addition to the completeness of the goal-driven specifications. Of course, one has to assume to be able to sufficiently specify a system by the description of its requirements. Although this is not true in general, this will be the case for the problem under consideration here. The major issue is to be able to say if, given a set of goals, and a measure of how the system meets them, the associated optimization problem is regular enough, or roughly speaking easy enough, such that an optimization method could succeed in finding an optimal solution. In other words, the main point is to able to design a measure such that a relevant optimization algorithm succeeds.

A Cellular Automaton (CA) is a discrete dynamical system composed by a set of finite state machines. A simple program, the transition function (see Section 2.1 for the details) defined the computation task of the CA. Although the CA is a simple computational model, it is a complex system that allows to solve a number of computational problems, and to model a wide range of physical and natural systems [3].

This work addresses a classical problem in the field of cellular automata, namely, the 6-states *firing squad synchronization problem* (FSSP) [4], which consists of synchronizing all the cells of the automata in the same state in minimal time. We associate FSSP with a discrete optimization problem, and provide a metaheuristic, a relevant Iterated Local Search, to solve it.

Moreover, we analyze the properties of the fitness landscape to explain the performances of our optimization algorithm in finding optimal solutions. Although the 6-states FSSP in optimal time has been already solved by Mazoyer in 1986 [5], in this work, by using an optimization technique, we found thousand of new solutions, some of which have a lower complexity than the original hand-made solution by Mazoyer, or present a different computational strategy.

There are a lot of previous works related to automated design of cellular automata by using metaheuristics, such as evolution-

*Corresponding author, Tel.: +33 3 59 35 86 30.
Email addresses: manuel.clergue@univ-ag.fr (Manuel Clergue), verel@univ-littoral.fr (Sébastien Verel), formenti@i3s.unice.fr (Enrico Formenti)

ary algorithms. Clearly our approach follows this trend.

The seminal work of Mitchell in 1993 is one the first studies which applied metaheuristics to cellular automata [6]. The authors used genetic algorithms to find efficient CA rules able to solve the density classification problem. This problem consists of being able to design a CA such that for any initial configuration c evolves to a specific target configuration if in c the density of symbols '1' is larger than the density of symbols '0' or not. Citing from [6]: *in general it is very difficult to program CA to perform complex tasks*, *genetic algorithms has promise as a method for accomplishing such programming automatically*. This statement can be easily generalized to metaheuristics.

In [7], Verel *et al.* gave a deeper insight in the fitness landscape structure of the search space associated to the majority problem, characterizing its neutral nature. Wolz *et al.* [8] proposed an efficient evolutionary algorithm to find solutions as good as possible for the majority problem. Note that, as noted by Oliveira in [9], the density classification problem has no exact solution.

Sapin [10] applied an evolutionary algorithm to find meaningful structures, glider guns, in a famous cellular automata, the so-called game of life. Still with cellular automata in 2 dimensions, Breukelaar in [11], and Aboud in [12], use evolutionary algorithms and metaheuristics to generate geometric shapes.

We conclude this short survey about metaheuristics applied to cellular automata search by pointing out the recent work of Bidlo where the author uses evolutionary algorithms to solve several CA problems [13]. Bildo's very interesting approach is based on conditional CA rules. However, this method is probably not adapted in our context since the number of rules is lower.

The next section is devoted to the precise definition of FSSP. The third section introduces the concept of fitness landscape. The fourth section explains how to turn the original FSSP into an optimization problem. The local search technique is exposed in the fifth section, and the sixth one contains the experimental results with the new solutions of 6-states as well as an analysis of the fitness landscape of the problem. In the last section we draw our conclusions and provide new perspectives.

2. Firing Squad Synchronization Problem

The Firing Squad Synchronization Problem (FSSP) was first proposed by Myhill in 1957 and published by Moore in 1964 [4]. This problem has been extensively studied for more than fifty years and has generated a large amount of literature. Indeed, it models one of the most important and fundamental problems in computer science: the decentralized synchronization of independent units. This section provides a precise definition of FSSP and a short survey of the main solutions.

2.1. Definition of the problem

The FSSP is defined over one-dimensional cellular automata. A (finite) one-dimensional cellular automaton (CA) consists of a set of n identical finite state automata (called *cells*) arranged on a line. States are chosen from a finite set Q called the set of *states*. The target size of Q is 6. Each cell computes its new state according to its own state and the one of a fixed set (the *neighborhood*) of neighboring cells using a *transition rule* (or simply *rule*) $\delta : \mathrm{Q}^k \to \mathrm{Q}$ where k is the size of the neighborhood. For our purposes, $k = 3$ and the neighborhood consists just in the cell itself and its left- and right-hand neighbors. All cells are updated synchronously at each time step.

Since the automaton is finite, special care should be taken for cells at the extremities of the line. Indeed, here cells at the extremities have just two neighbors and not three.

A *configuration* is a mapping from the interval of integers $[\![1, n]\!]$ to Q which gives the state of each cell at a given time step. A transition rule $\delta : \mathrm{Q}^3 \to \mathrm{Q}$ induces a global rule $G_\delta : \mathrm{Q}^{[\![1,n]\!]} \to \mathrm{Q}^{[\![1,n]\!]}$ as follows

$$\forall c \in \mathrm{Q}^{[\![1,n]\!]}\ \forall i \in [\![1, n]\!]\ \ G_\delta(c)_i = \delta(c_{i-1}, c_i, c_{i+1})$$

(recall that the extremities have only two neighbors).

The *space-time diagram* of initial configuration c is a graphical representation of the orbit of c and consists of the superposition of c, $G_\delta(c)$, $G_\delta^2(c)$, etc. Usually, a space-time diagram is drawn in increasing time steps from the top to bottom (see Fig.1). Finally, remark that a (finite one-dimensional) CA is completely characterized by its transition rule and in our case can be represented by a three entries table.

In order to better illustrate the FSSP, we assign a special name to states in Q. Indeed, G is the *general* state, F is the *firing* state, $\bullet$ is the *quiescent* state and, finally, \$ is the *border* state. Remark that the quiescent state must satisfy:

$$\delta(\bullet, \bullet, \bullet) = \delta(\$, \bullet, \bullet) = \delta(\bullet, \bullet, \$) = \bullet$$

An admissible initial configuration c for FSSP is such that $c_1 = G$ and all other cells are in quiescent state. An admissible final configuration c for FSSP is such that all cells are in firing state.

FSSP
INSTANCE: a set of states Q
SOLUTION: a transition rule $\delta : \mathrm{Q}^3 \to \mathrm{Q}$ such that for any admissible initial configuration $c : [\![1, n]\!] \to \mathrm{Q}$ there exists an admissible final configuration $d : [\![1, n]\!] \to \mathrm{Q}$ and $t \in \mathbb{N}$ such that $G_\delta^t(c) = d$ and $G_\delta^{t'}(c)_i \neq F$ for all $i \in [\![1, n]\!]$ and $t' < t$.

In other words, solving FSSP means providing a transition rule δ such that for any admissible initial configuration, it synchronizes all the cells in firing state and no cell is in firing state before the final admissible configuration.

Note, that the minimal time required to synchronize all cells in the firing state for the first time is $2n - 2$ for configurations of size n. In this work, we are interested in finding solutions to FSSP which synchronize in minimal time and have 6 states.

The main difficulty is to find such transition solutions regardless of the length n of the line with the same set of states. Fig. 1 shows an illustration of the problem.

Finally, during the CA evolution until an admissible final configuration or the maximum time step is reached, entries in

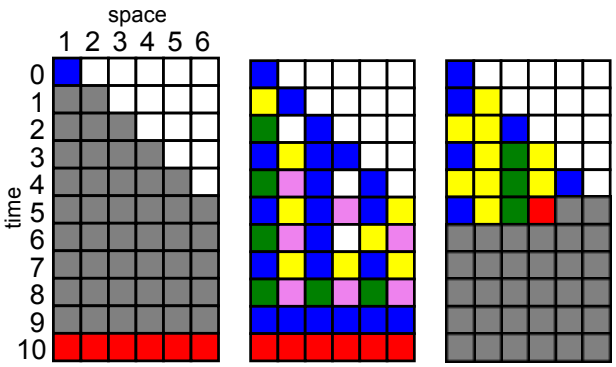

Figure 1: Space-time diagrams of the FSSP for a line of length $n = 6$ in minimal time $2n-2$. The quiescent state is white, the initiator state (general state) is blue, and the firing state is red. Grey cells are still undefined. On the left, the space-time diagram is an illustration of the problem with the initial configuration and the expected final state. In the middle, one example of transition function that synchronized correctly in minimal time. On the right an example of rule which failed to synchronize. Indeed, one cell is firing before the other ones.

the transition table are never used, we call them *unused* entries. On the other hand, the *used* entries are those entries of a transition rule that are used at least once while computing an evolution from an admissible initial configuration.

2.2. *Known solutions*

The FSSP was the subject of a abundant literature both in its original form and in its numerous variants. The search for optimal time solutions using a minimal number of states is one of the central issues of the research about FSSP.

The first official solution for the FSSP was given by Minsky and McMarthy in 1967 [14]. It uses 15 states and synchronizes a line of size n in $3n$ time steps. They implemented a divide and conquer strategy which was at the basis of several subsequent solutions by other authors.

The first solution in optimal time (that is $2n-2$) was found by Eichi Goto even before the well-known solution of Minsky and McMarthy. However, Goto's implementation uses several thousand of states [15] and remained mythical, in some sense, for several years until some researchers had the manuscript directly from the author.

Few years after Goto's solution, Waksman [16] and Balzer [17] proposed optimal time solutions with 16 and 8 states, respectively.

In 1986, Jacques Mazoyer [5] designed a solution in minimal time which uses only **6** states. It is the current best solution with minimal time and with the lowest number of states. Fig. 2 shows a space time diagram of this clever hand-made solution with only 6 states. Like previous solutions with a higher number of states, the solution of Mazoyer also uses a divide-and-conquer strategy, but it divides the original problem of size n into a sub problems of size one third of this length: $(2/3)n$, $(2/3)^2 n$, etc.

In [18], Jean-Baptiste Yunes introduces the notion of Kolmogorov complexity of a FSSP solution as the number of transitions used by the solution. Mazoyers's solution has complexity 119. From this point of view, the current record is held by Umeo's solution with complexity 78 [19] but its time complexity is $3n$. If one turn to optimal time complexity, then the current record is 80 and it is one of the contributions of the present paper (see Section 6.2).

Balzer [17], and later Sanders [20], proved that there is no optimal time solution for 4 states able to synchronize all the lines for any n. Remark that there are solutions which synchronize lines of length 2^n but not lines of any length [21].

The existence of a 5-states solution still remains a major open question in this subject, even in non-minimal time.

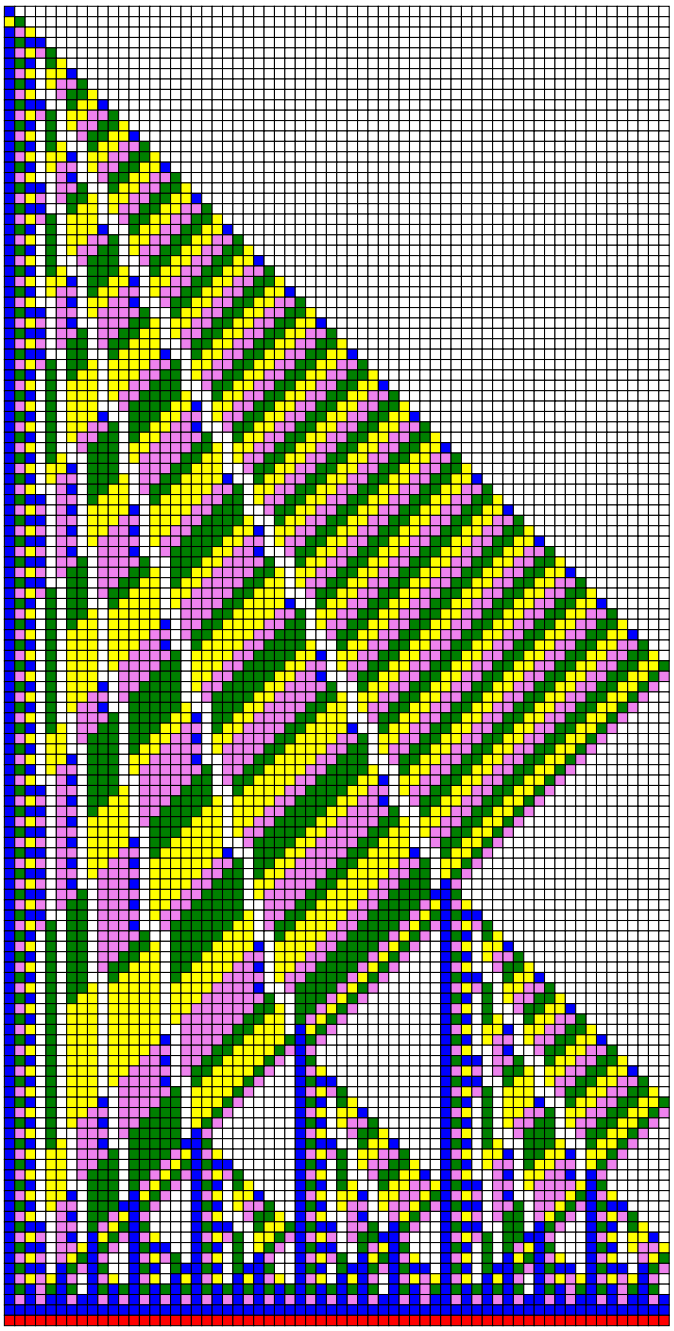

Figure 2: Space-time diagram of the Mazoyer's solution for $n = 64$.

3. Fitness landscape

The fitness landscape of an optimization problem is a triplet $(\mathcal{X}, \mathcal{N}, f)$ where $\mathcal{X}$ is the set of all possible solutions, $\mathcal{N} : \mathcal{X} \rightarrow 2^{\mathcal{X}}$ is the neighborhood relation between solutions, and $f : \mathcal{X} \rightarrow \mathbb{R}$ is the fitness function to maximize (for more details see [22], for instance).

The structure of the fitness landscape deeply impacts the performance of a local search algorithm such as Iterated Local Search (ILS) [23, 24]. This section introduces the two mains geometries of fitness landscapes which will useful to deeply analyze the results obtain for the FSSP.

3.1. Multi-modality and Ruggedness

One of the main features of the fitness landscape is related to local optima. A *local optimum* $x^\star$ is a solution such that for every solution in the neighborhood, no strictly better solution exists: $\forall x \in \mathcal{N}(x^\star), f(x) \leqslant f(x^\star)$. Several works demonstrate the link between the performance of local search and the shape of the fitness landscape. For example, the number of local optima and the network of basins of attraction influence the probability of reaching the global optimum [25].

Ruggedness is another important feature which is linked to the local continuity of the fitness landscape. The search is as difficult as the more rugged the landscape it is. The autocorrelation of fitness during a random walk (*i.e.* the correlation between solutions separated by several random steps) is a classical way to measure ruggedness [26].

The autocorrelation of fitness is useful to compare different fitness landscapes and it can be used to predict performance. However, autocorrelation is not relevant to study precisely the ruggedness of one particular fitness landscape, as the scale of the correlation value is not easily interpretable alone without any comparison. In this work, we prefer to use the *fitness cloud* which gives a better insight into the correlation of fitness between neighboring solutions [27]. The fitness cloud is the conditional bivariate probability density of reaching a fitness value from a solution of a given fitness value applying a local search operator [7]. Several scatter plots and metrics can be deduced from the fitness cloud. The average fitness of solutions in the neighborhood according to the fitness value of the solution gives a global picture of the correlation. When the average fitness in the neighborhood is independent of the fitness value of the solution, the fitness landscape is highly rugged, and one cannot expect to optimize the problem using a local search based on this neighborhood relation. On the contrary, when the average is close to the fitness of the original solution, a local search should have a higher probability to progressively find better solutions.

3.2. Neutrality

Neutrality was originally introduced in the context of molecular biology [28]. It is another major global geometry of the fitness landscape. Intuitively, a neutral fitness landscape is dominated by large plateaus. In those particular landscapes, search dynamics are mainly explained by random mutations over plateaus accepting neighboring solutions with the same fitness value. This kind of dynamics is characterized by alternance between rapid increases of the fitness and long stagnations of fitness over the plateaus. Moreover, during the neutral exploration of the search space, the search performs a random exploration, and is pushed toward solutions with higher neutral degree [29]. The *neutral degree* of a solution is defined as the number of neighboring solutions with the same fitness value [30]. The statistics related to neutral degrees (average, distribution, network statistics, etc.) are the major metrics to describe neutral fitness landscapes.

Some optimization problems related to cellular automata are already known to be shaped as a neutral fitness landscape [6, 7].

The FSSP fitness function will be designed to give a minimal, but non misleading, information on the quality of the solution during the search. Hence one can hypothesize that the fitness landscape of the FSSP optimization problem is also a neutral one. Deciding if the introduction of neutrality into an optimization problem is a drawback or an advantage, it is still an open question. Neutrality can be interpreted as a lack of information that enables to guide the search, or a useful technique to remove local optima which drives the search in wrong directions [31, 32]. In any case, a dedicated local search algorithm must be designed when the fitness landscape is neutral such as the *netcrawler* [33], the scuba search [34], or the NILS [35]. As a consequence, we use the *netcrawler* of Barnett [33] which accepts neighboring solutions with the same fitness value (see lines 8 and 12 of Algo. 1). The neutrality of the fitness landscape and the efficiency of a dedicated algorithm for FSSP will be analyzed in the next sections.

4. The FSSP as an optimization problem

Our goal is to associate an optimization problem to the FSSP in such a way that an optimal solution of the first is a solution of the second. The definition of an optimization problem involves to define two elements: the search space which is the set of potential solutions, and a fitness function which measures the quality of solutions. Endowing the optimization problem with a neighborhood relation between potential solutions defines the *fitness landscape* where the search dynamics occurs.

The construction of the fitness landscape is crucial at the extent that the choices made will influence the research process and thereby the success of the search for optimal solutions. As we are not trying to find an approximation of an optimal solution of the optimization problem, but a solution of the initial FSSP problem, it is certain that the choices we make are even more crucial. There is a trend, justified by experience and by the theory, which is to incorporate as much information and knowledge about the problem in the definition of the fitness landscape. It is important to guide the search, for example by eliminating the symmetries in the representation of potential solutions [36], by using a fitness function which could guide progressively towards the good solutions [37] or by considering a suitable neighborhood relation [38]. Due to the lack of thorough knowledge about this problem from the point of view of local search optimization, we decided to do the opposite of that principle by designing the elements of the landscape

in the most direct way possible with a minimal expertise, although some properties of the FSSP are still taken into account. In such a way, we try to not introduce misleading information which could make the optimization problem more difficult (by introducing local optima, for example).

4.1. The search space

The search space $\mathcal{X}$ of the optimization problem associated with the FSSP consists of the set of all possible CA transition functions. In the case of the 6-states FSSP, the set of states is augmented to include a *border* state so that we have $Q' = \{\bullet, G, B, C, D, F, \$\}$. In particular, $\bullet$ is the quiescent state, G is the initiator state, F the firing state, and \$ is the border state. Not all transition functions based on Q' are in the search space. For example the border state can not appear in the middle of transition rules. Moreover, the computation of the target CA stops when a cell reaches a firing state, so the firing state never appears as the antecedent of transition rules. Also, when considering the 3 transition rules of the quiescent state given by the definition of the FSSP, the maximum number of possible transition rules is $5^3 + 2 \times 5^2 - 3 = 172$. Hence, the size of the search space is at most 6^{172}.

Some basic observations of previous FSSP solutions proposed by Jacques Mazoyer and other authors lead to reduce the search space size. First, one can assume that a firing state appears in the output of transition rules only when all neighboring cells and the cell itself are in the initiator state: $\delta(G, G, G) = F$, $\delta(\$, G, G) = F$, and $\delta(G, G, \$) = F$. This basic hypothesis reduces the size of the search space by a huge factor, and probably increases the density of "good" potential solutions. For cells line of length 2, it is very easy to find transition rules to ensure the synchronization. We fix 4 transition rules in the same way as Mazoyer's solution (see Fig. 3 left for an illustration): $\delta(\$, G, \bullet) = B$, $\delta(G, \bullet, \$) = B$, $\delta(\$, B, B) = F$, and $\delta(B, B, \$) = F$. Taking into account those simple hypothesis, the number of transition rules of the search space becomes $172 - 7 = 165$ possible transitions, and the size of the search space is now $|\mathcal{X}| = 5^{165}$. In summary, the set of the 165 antecedents of transition rules in the search space is:

$$
\begin{aligned}
A_{rules} = \quad & ((Q \setminus \{F\})^3 \cup (\{\$\} \times (Q \setminus \{F\})^2) \cup ((Q \setminus \{F\})^2 \times \{\$\})) \\
& \setminus \{\ (\bullet, \bullet, \bullet), (\$, \bullet, \bullet), (\bullet, \bullet, \$), \\
& \quad (G, G, G), (\$, G, G), (G, G, \$), \\
& \quad (\$, G, \bullet), (G, \bullet, \$), (\$, B, B), (B, B, \$)\}
\end{aligned}
$$

Of course, the size of the search space is still so huge that it is impossible even for modern high-performance computers to fully enumerate the whole search space.

4.2. Fitness function

The fitness function associates a real number with each potential solution. This real number represents the quality of the transition function. In short, the quality is the length of the largest line which is correctly synchronized by the transition function. More formally, for any integer $i \geqslant 2$, the decision problem fssp_i is true when the cells line (configuration) of length i is correctly synchronized according to the definition of the FSSP. The fitness value of a solution is equal to the largest synchronized length such that all lines with a shorter length are also synchronized: for any solution $x \in \mathcal{X}$, $f(x) = n$ iff $\forall i \in [\![2, n]\!]$, $\text{fssp}_i(x) = \text{true}$ and $\text{fssp}_{n+1}(x) = \text{false}$. The definition of the search space ensures that the fitness value of all solutions is greater than 2. When $\forall i \geqslant 2$, $\text{fssp}_i(x) = \text{true}$, then, by convention, the fitness value of the solution x is infinite *i.e.* $f(x) = \infty$. Fig. 3 shows an example of fitness value computation.

The optimization problem associated with the FSSP is $\text{FSSP}_{opt} = (\mathcal{X}, f)$, where $\mathcal{X}$ is the search space defined in Section 4.1, and where f is the previously defined function which measures the quality of each solution. If the maximum value of the fitness function f is ∞, then the FSSP is solved.

In order to reduce the computation time when the solution has a possibly infinite fitness value, the CA computation is stopped when a line of length $n = 100$ is synchronized. This bound could be misleading to detect solutions of the FSSP since some solutions with fitness value 100 does not synchronize lines with a larger length, and thus are not solutions of the FSSP. However those false positive solutions of the FSSP are few in number, as we will see in the experimental results reported in Section 6.

The computation time of the fitness values can be reduced using the properties of the FSSP. For a line of length i, there are $i(2i-2)$ cell values to assign. It is not necessary to compute the value of the whole cells using the transition function. Indeed, the 3 rules on the quiescent states leave in quiescent state the upper right triangle of cells in the space-time diagrams. Hence, the number of cells to compute reduces to $i(3i-1)/2$. Moreover, it is possible to check the correction of the synchronization before the last time step. Based on the rules which output a firing state, when the transition function correctly synchronizes the line, the state of cells in the penultimate line in the space-time diagram can only be the state G (except for the 2 cells at left and right ends of the line for which the state could also be B). It reduces by a gap of n the number of cells to compute. At last, a more tricky property can be used. The state of cells in the large upper left part of space-time diagram can be deduced from the state of cells of the previous length $i-1$ (see Fig. 4). The first cell for which the state of right neighbor is modified in comparison of the previous length is the cell number $i-1$ (before the last cell) at time step $i-2$. Finally, for a line of length i, at most $i(i+3)/2$ cells are computed to evaluate the fitness value of a solution. From a very basic computation method, it reduces the computation complexity by a factor 4 allowing the exploration and the evaluation of a larger number of solutions during the optimization.

5. Optimization algorithms

In this work, we wanted to focus on the relationships between the geometry of the search space and the problem rather than the advanced local search techniques. So we chose an Iterated Local Search (ILS) method combined with a hill-climbing

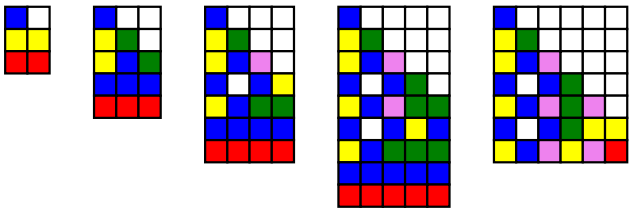

Figure 3: Space-time diagrams of a solution of fitness value equal to 5 which is the largest synchronized line by this solution.

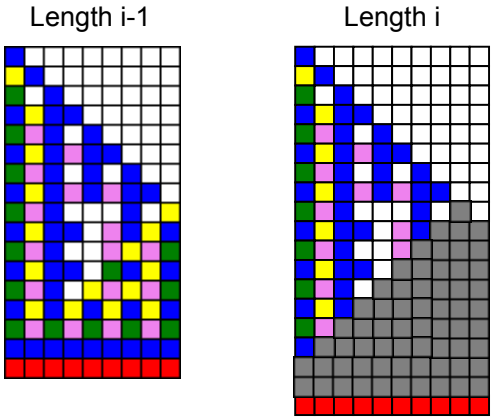

Figure 4: Efficient evaluation: only cells in grey have to be computed between line of length $i-1$ and i.

algorithm, in order to highlight and exploit the local properties of the search space.

5.1. Hill-climbing

Hill-Climbing algorithms (HC) are simple, but efficient, algorithms for combinatorial optimization. This class of algorithms often takes part in state-of-the art algorithms for solving challenging optimization problems [39, 40]. HC algorithms follow the idea of gradient search by continuously improving the current solution. Starting from an initial solution, at each iteration, a set of neighboring solutions is evaluated and one improving solution is selected according to a pivot rule for the next iteration of the algorithm. Usually, HC search is stopped when no improving solution can be found (local optimum), or when either the quality of the current solution is sufficient, or the computational budget is spent. The choice of the neighborhood relation between solutions and of a relevant pivot rule are the key points of HC design.

The neighborhood relation can be defined using a distance between solutions, *i.e.* the neighbors are then defined by the set of solutions at a distance less than a given constant, or by an elementary move which defines the way to reach a neighboring solution. In this work, the distance D between two solutions is defined by the number of transition rules with distinct images:

$$D(\delta_1, \delta_2) = \sum_{(\ell,c,r)\in A_{rules}} [\delta_1(\ell, c, r) \neq \delta_2(\ell, c, r)] \qquad (1)$$

where $[a \neq b] = \begin{cases} 1 & \text{if } a \neq b \\ 0 & \text{if } a = b \end{cases}$ (Iverson bracket notation). The neighborhood is defined by the set of solutions at distance 1: for any solution $x \in \mathcal{X}$, $\mathcal{N}(x) = \{y \in \mathcal{X} : D(x, y) \leqslant 1\}$. A neighboring solution is obtained by the modification of one transition rule. The number of possible transition rules is $|A_{rules}| = 165$ (see Sect. 4.1), and the state in the consequence of the rule can be changed to 4 other possible different states, so the size of the neighborhood is $4 \times 165 = 660$.

A pivot rule selects a solution from a set of evaluated neighbors. Among a large number of possible choices, two main variants of the pivot rule in HC algorithms are commonly used. The *best-improvement* HC selects the best neighbors, *i.e.* one neighboring solution with the best fitness value which implies to evaluate all solutions in the neighborhood, and the *first-improvement* HC selects at random one improving neighbor even if it does not have highest fitness value among the neighbors, *i.e.* the first random neighbors which improves the current solution is selected. The choice of the pivot directly impacts the performance of HC, either in term of solution quality, or in term of the computational cost. Recent works [40, 41] highlight that first-improvement tends to be more efficient on most landscapes. The Algo. 1 shows the first-improvement HC used to solve the FSSP optimization problem, denoted as *neutral HC* (NHC). Indeed, we pay particular attention to the accepting condition (line 10), where the algorithm moves even when the neighbor has the same fitness value. Due to the limited number of fitness values and the large size of the search space, we suspect that the fitness landscape is neutral and dominated by large plateaus of neighboring solutions with the same fitness value. The accepting condition ensures that the search will drift on neutral plateaus. This variant of first-improvement HC dedicated to neutral landscapes is also denoted as Netcrawler [33]. However, the algorithm is not able to stop naturally when a local optimum is reached, the search could potentially drift indefinitely on plateaus. So, a stopping criterion is added: when the algorithm reaches a maximum number of evaluations e_{hc}, it stops.

Algorithm 1 First-improvement hill-climbing with neutral acceptance criterion (NHC).

1: **Inputs:** x : initial solution,
2: e_{hc} : maximum number of evaluations
3: Evaluate x using f
4: $e \leftarrow 1$
5: $S \leftarrow \mathcal{N}(x)$
6: **repeat**
7: Choose a neighbor randomly $y \in S$
8: $S \leftarrow S \setminus \{y\}$
9: Evaluate y using f, and increment e
10: **if** $f(x) \leqslant f(y)$ **then**
11: $x \leftarrow y$
12: $S \leftarrow \mathcal{N}(x)$
13: **end if**
14: **until** $S = \emptyset$ or $e \geqslant e_{hc}$
15: **Output:** x : best solution found

5.2. Iterated Local Search

The Iterated Local Search [42] is an algorithm which combines a simple local search algorithm with a perturbation mechanism to restart the search in a more promising zone of the search space. An interesting review on ILS can be found in [43]. The Algo. 2 shows the ILS algorithm defined in this work. The simple local search is the first-improvement hill-climbing described in Sect. 5.1.

To restart the HC algorithm, the current solution is modified by the perturbation procedure. The perturbation modifies a number of transition rules in the solution, but operates differently according to the nature of the transition rule. For all unused transition rules in the solution (see def. in Sect. 2.1), *i.e.* the transitions which do not contribute in the CA computation, the state in the rule output is modified at random. Obviously, changing unused rules only has no effect on the fitness of the solution, but allows to perform large steps in the search space without losing quality, and hence potentially it permits to find a new interesting sub-space to explore. In addition to the previous modifications, the perturbation also modified k used transition rules at random. The parameter k is the strength of the perturbation. Intuitively, when the strength k is too small, the search is around the current solution, and it should more difficult to find a better solution after the HC; when the strength k is too large, the search restarts from a random solution away from the current sub-space which contains the best current known solution. In practice, the strength k must be tuned according to the fitness landscape.

Lastly, the ILS starts from a random solution of the search space $\mathcal{X}$. No particular heuristic is used to initialize the algorithm. Basically, the ILS stops when the computational budget is spent, *i.e.* when the number of evaluations reaches the parameter e_{max}.

Algorithm 2 Iterated Local Search (ILS)

Inputs: e_{max} : maximum number of evaluations
  e_{hc} : number of iterations in each hc
  k : perturbation strength between two hc
Choose randomly an initial solution $x \in X$
$x \leftarrow \text{HC}(x, e_{hc})$
Initialize the number of evaluations e_{tot}
repeat
  $y \leftarrow \text{perturbation}(x, k)$
  $z \leftarrow \text{HC}(y, e_{hc})$, and update the number of evaluations e_{tot}
  if $f(x) \leqslant f(z)$ **then**
    $x \leftarrow z$
  end if
until $e_{tot} \geqslant e_{max}$
Output: x : best solution found

6. Experimental results

6.1. Performance of ILS

This section describes the results obtained by the ILS algorithm applied to the FSSP optimization problem. We will analyze the different components and parameters of the algorithm, and their influence on its performances.

In Section 5, we saw that the local search algorithm embedded by the ILS is a modified first-improvement hill climbing where the acceptance criterion is adapted to neutral landscapes. Our first experiment aims to show that this choice is efficient. Figure 5 shows the performance of hill-climbing algorithms with different acceptance criteria. HC with strict improvement (SHC) only accepts neighboring solutions with a strictly higher fitness value. So, this algorithm stops when all neighbors have lower or equal quality. The average fitness found is very low, around 2.12. Most of the runs were not able to improve the initial solution of fitness 2. Of course, the number of evaluations is also very small, on average only 700 solutions are evaluated. In order to compare the different HC with the same number of evaluations, SHC is embedded into a ILS-like framework with a restart strategy. When SHC stops, it is restarted either from a random solution (shc_r), or from a solution where 3 transitions rules has been modified at random (shc_3). The hill-climbing of algorithm 1 with loose acceptance criterion is called neutral hill-climbing and denoted *nhc*. For all HC algorithms, the maximum number of evaluations e_{max} is set to 5×10^6. Although the strict improvement HC with random restart has the same number of evaluations, the average performance is only 4.80. In comparison, the *nhc* used in the ILS is 14.81. The average performance of the shc_3 is 10.85, again much lower than the NHC. The acceptance of tie neighboring solution is an important feature of the algorithm: it is easier to find an improving solution from a local optima and from neutral moves rather than from a random solution. The acceptance of equal fitness solutions has a positive impact on the quality of solution found by the HC.

The ILS algorithm (see algorithm 2) has 2 main parameters: e_{hc}, the number of evaluations for each hill-climbing and k the strength of the perturbation, that is the number of used transition rules which are modified. Table 1 shows the number of successful runs out of the total of 200 runs according to the two parameters e_{hc} and k. Except when the number of success is 0 or 1, according to the Fisher's Exact test at confidence level of 5%, there is no significant difference of performance between all pairs of parameters. For the same total number of evaluations (100×10^9), the number of evaluations in each HC and the perturbation strength does not strongly impact the performance of the ILS algorithm except for the extremal values. For a large number of evaluations in each HC (5×10^6), or large perturbation strength ($k \geqslant 6$), the performance is deteriorated according to Fisher's Exact test at confidence level of 5%.

Two experiments, not shown in the table, tend to comfort the hypothesis of the utility of the perturbation, with a low value of k. For both of them, we set e_{hc} to 1.1×10^6. The first one simulates a random restart, with k set to 120. None of the 200 runs found the optimal solution (the maximum fitness value is 46). The median is 30, that is much lower than the case with $k \in [\![2, 7]\!]$. On the opposite, for the second one, we set k to 0. Again, none of the 200 runs were able to find the optimal solution (the maximum fitness value is 45). The median is 18, that is much lower than when $k \in [\![2, 7]\!]$.

Figure 6 shows the distribution of the best fitness found for

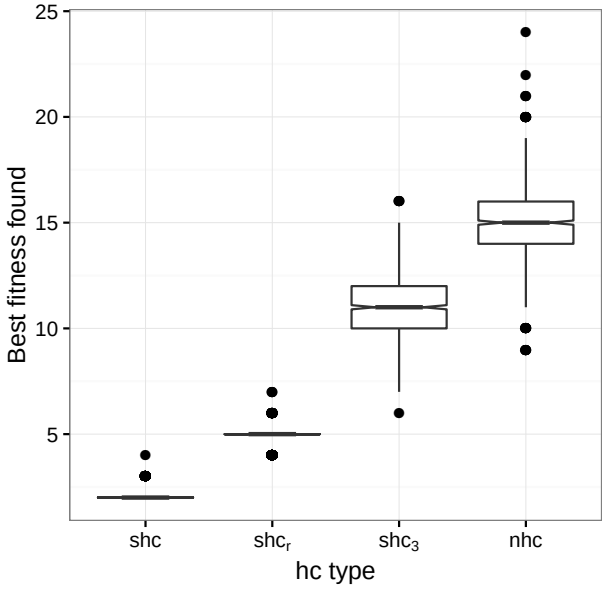

Figure 5: Boxplot of best fitness found for different hc variants: HC with strict improvement (*shc*), HC with strict improvement and restart (shc_r), HC with strict improvement and restart at maximum distance 3 (shc_3), and HC with neutral acceptance (*nhc*). Each one is run 1000 times for 5×10^6 fitness evaluations (except strict improvement which converges quickly). This shows the interest of neutral acceptance, since the hc that implements it clearly outperforms the others.

Table 1: Number of successful runs which find an optimal solution on the 200 runs for each couple of parameters (number of evaluations in each HC (e_{hc}) and strength of the perturbation which modifies k transition rules. Cumulative numbers are given for each parameter value on lines and columns. Statistical significant best success rates according to Fisher's Exact test at confidence level $p = 0.05$ are in bold.

hc eval e_{hc} ($\times 10^6$)	Perturbation k						
	2	3	4	5	6	7	cumu.
0.3	**7**	**3**	**4**	**4**	**5**	**2**	**25**
0.5	**6**	**7**	**4**	**5**	**5**	**3**	**30**
0.7	**3**	**8**	**8**	**8**	1	**6**	**34**
0.9	**2**	**9**	**8**	**5**	**2**	**2**	**26**
1.1	**6**	**8**	**6**	**4**	**7**	0	**31**
1.5	**7**	**9**	**3**	**5**	**5**	0	**31**
5.0	**3**	**3**	**5**	1	0	**3**	15
cumu.	**34**	**47**	**38**	**32**	25	16	192

each setting. According to the Mann-Whitney statistical test at confidence level of 5%, there is no significative difference of performance between 0.7×10^6, 0.9×10^6, and 1.1×10^6 number of evaluations in hill-climbing whereas low and high numbers of evaluation in hill-climbing reach a lower median of fitness value. However, for the best numbers of evaluation in hill-climbing, there is no significative difference of fitness between the strength of the perturbation. We also check the number of evaluations to reach an optimal solution (not presented here), but again there is no statistical difference between parameters. The experiments have been run on a cluster of heterogeneous machines, so computational time is not very significative in our case. However, the computational time is always proportional to the number of evaluations such as the other part of the optimization algorithm is computationally inexpensive. In the following, we comment the number of evaluations to reach an optimal solution. Overall, the average number of evaluations is 50.04×10^9 evaluations to reach an optimal solution. This is approximately the half of the total number of evaluations. The standard deviation is 27.06×10^9 which is high compared to the mean value.

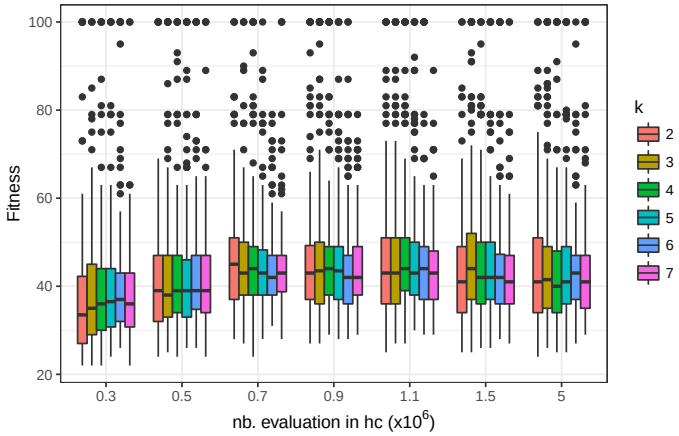

Figure 6: Boxplot of best fitness found by ILS according to the number of evaluations in hill-climbing (HC) and the strength of the perturbation (k).

Indeed, the success rate depends on the total number of evaluations. Globally, the average success rate on all experiments is 2.29% with 100×10^9 evaluations. Figure 7 shows the average success rate of all ILS according to the number of evaluations. No optimal solution is found before 10^9 evaluations. Surprisingly, the success rate increases linearly with the total number of evaluations. On average, the success rate increases by a step of 0.024% for 10^9 evaluations between 10^9 and 10^{11} evaluations.

6.2. Analysis of the optimal solutions

The ILS algorithm does not stop when an optimal solution is found. It can continue by applying on the solution the perturbation operator and then by the first-improvement hill-climbing *NHC*. Remember that to limit the computation cost, the computation of the fitness is bounded to 100, even if the fitness of the solution is greater. From a single successful run, it is possible to find several different solutions with maximum fitness value 100. In order to confirm the optimality of such solutions and to remove the duplicated ones, we perform a post-processing on the solutions until a cells line of length $n = 2 \times 10^3$, and we also remove the solutions which have exactly the same space-time diagrams, *i.e.* which have the same used transition rules. Although we can not formally prove that a solution with a fitness $n = 2 \times 10^3$ is an optimal solution which

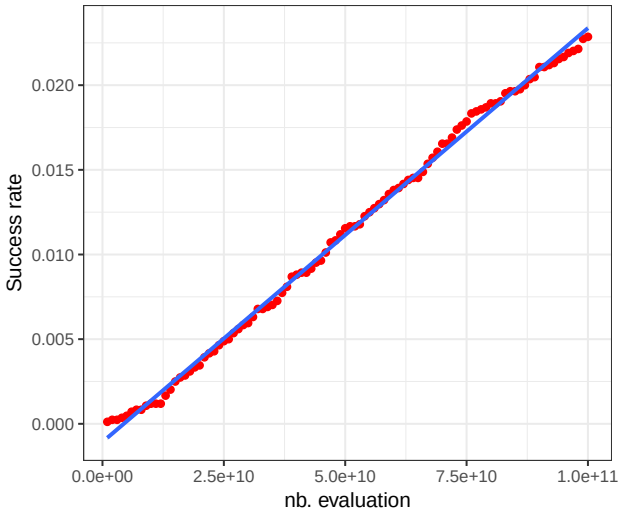

Figure 7: Average success rate of ILS algorithms, and the regression line according to the number of evaluations.

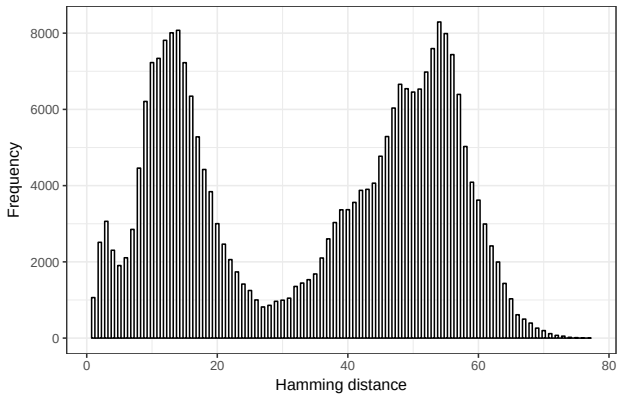

Figure 8: Distribution of the Hamming distance between the 718 optimal solutions.

synchronizes all cells lines of any length, only 121 unique solutions have a fitness greater than 100 but strictly smaller than 2×10^3 which represents $14,42\%$ of the total number of solutions with fitness 100. In addition, the greatest fitness strictly lower than 2×10^3 is only 147 which is much lower than 2×10^3. Moreover, we also consider the symmetry between states. The quiescent state $\bullet$, the initiator state A, the firing state F, and the state B used for synchronization of length $n = 2$ are particular states, and are not exchangeable. But the states labelled C and D can play the same role. We also check this symmetry between states, and consider that two solutions are the same if the permutation of C and D gives the same set of space-time diagrams.

From all experiments composed by $8,400$ runs, 718 different optimal solutions have been found from the 192 successful runs. This result gives a large set of optimal solutions that can be analyzed[1].

We define the Hamming distance between two solutions by the number of used transition rules which are different. Indeed, one transition rule can be used in the computation of one solution, but not in the other one. So, the transition rule of the solution which does not use this rule is free, and we consider that the two solutions does not differ on this rule. Fig. 8 shows the distribution of the Hamming distance between each pair of optimal solutions. Only 0.41% pairs of solutions are at Hamming distance 1, so few solutions are at the minimal distance for which they differ only on one transition rule. The maximum distance is 77, and the average distance between solutions is 34.58 which is large compared to the average number of used transition rules. The distribution is dominated by two main peaks centering respectively at Hamming distance 13, and 54. The height of the peaks are nearly equal. However, the second peak around distance 54 is more spread than the first one. This result suggests that at least two different clusters of optimal solutions have been found, one of the cluster could be larger than the other one.

Fig. 10 shows the histogram of the number of used transition rules for the set of optimal solutions. The number of used transition rules is the size of the program of the cellular automata, and then it is the Kolmogorov complexity for CA [18]. The optimal solution in minimal time of Mazoyer's uses 119 transition rules. In our set of optimal solutions, on average 105.3 transition rules are used, and the largest complexity is 127 transition rules. However, the distribution is not unimodal. The Kolmogorov complexity of the optimal solutions found by the ILS is much smaller than the original Mazoyer's solution. Indeed, from the 718 solutions, 665 optimal solutions (92.5%) use strictly less than 119 transition rules. The way of exploring the CA search space is probably very opportunistic. The search starts with solutions using a small number of transition rules that can solve the FSSP with short lines of automata. Then, we hypothesis that the solutions with a smaller number of used transition rules, i.e. with a smaller complexity, are easier to find than solutions with a larger number of used transition rules with a larger complexity.

We discovered 2 optimal solutions which use only 80 transition rules[2]. This constitutes a new record for the 6-states FSSP in minimal time. In comparison, the Umeo's solution [19] which has the minimal Kolmorogorov complexity uses 78 transition rules. Indeed, it is a 6-states solution but non in, minimal-time (time complexity is $3n+O(\log n)$). The 2 new solutions are very close to the minimum Kolmogorov complexity of Umeo's solution but they are in minimal-time. Fig. 9 shows an example of space-time diagram for a line of length 64 of one of the optimal solution using 80 transitions.

Fig. 11 shows the number of used transition rules according to the quality of the solutions. The sample of solutions is not

[1] All new solutions of the 6-states FSSP and the tools to visualize them will be available on a web page of the authors if the paper is accepted.

[2] Indeed, the transition rule $\delta(\$, \bullet, \bullet) = \bullet$ is not used. So, only 79 transition rules are really used. But still, we count 80 transition rules such as this rule is mandatory by the definition of the FSSP.

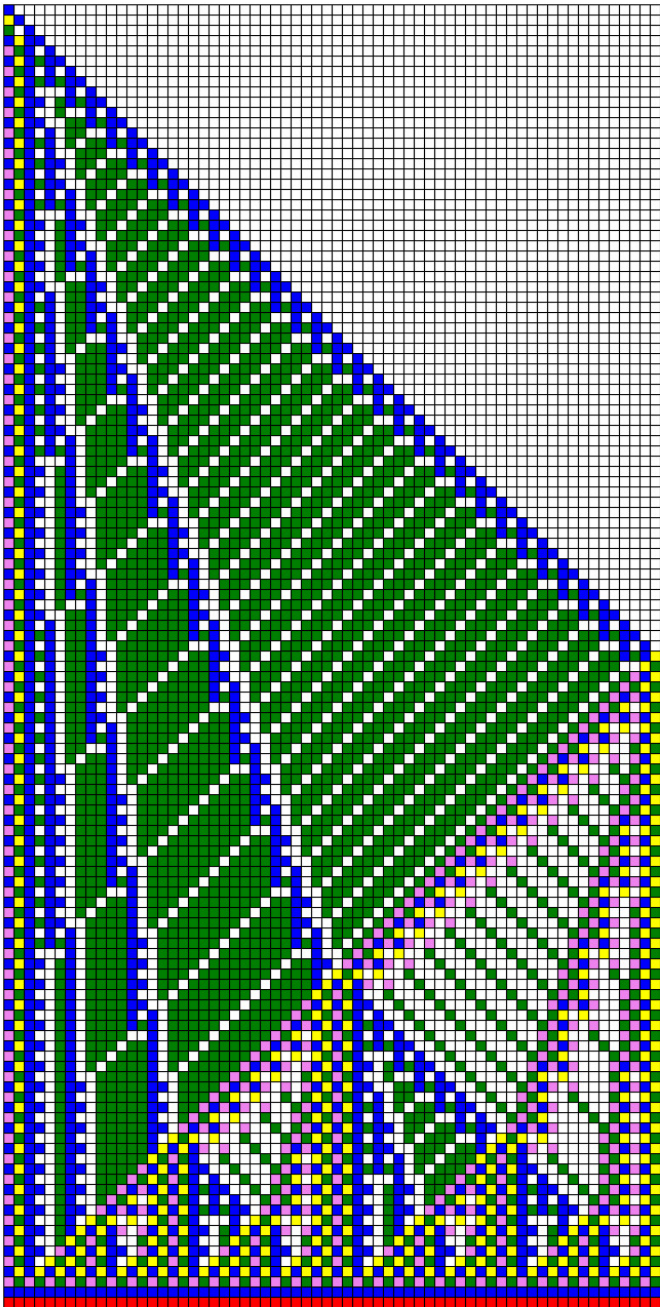

Figure 9: Space-time diagram (cells line of length 64) of an optimal solution with the minimal Kolmogorov complexity for a minimal-time solution. This solution uses only 79 transition rules, but we considered this solution has a Kolmogorov complexity of 80 (see the footnote 2).

random, it is composed by the local optima found along the different runs of ILS. For each run, the first solution found of a given fitness value is recorded into the sample. Overall when duplicated solutions are removed, the sample size is $72,991$. For low quality solution, the number of used transition rules is small. It increases with the fitness value of the solution until the length 18, then it becomes more or less constant around 122 used rules. The complexity of CA increases with the difficulty of the task, *i.e.* the length of the line of automata. On average, the search starts with solutions of low complexity, then it becomes progressively necessary to use more transition rules in order to optimize the problem. However, the search tends to find optimal solutions with a lower complexity compared to Mazoyer's strategy.

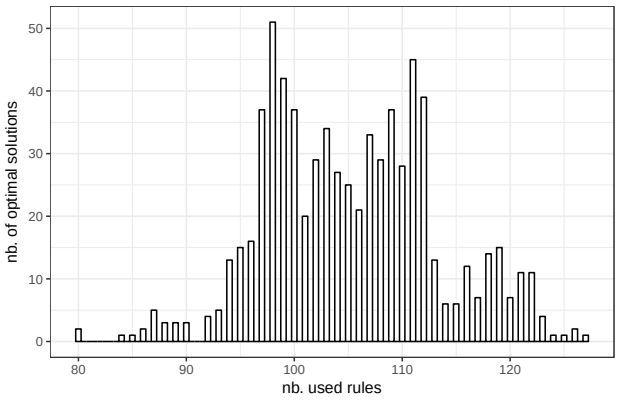

Figure 10: Distribution of number of used rules in the set of optimal solutions.

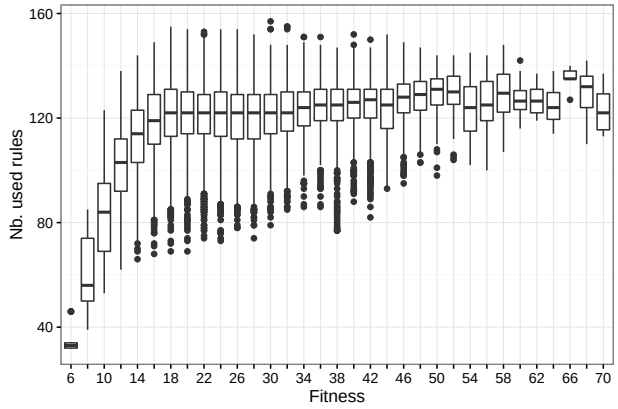

Figure 11: Boxplot of number of used rules according to the fitness.

Mazoyer's solution is a recursive solution sending an infinite number of signals to divide the initial problem into subproblems of size one third of the original size. In order to compare the strategies of the new optimal solutions, they are classified according to the upper left triangle in space-time diagram. For a given line of length n, we define the distance D_n^{up} by the number of different cells in the upper left triangle of the space-time diagrams of length between 2 and n. More precisely, the distance d_p^{up} for a line of length p between two solutions x_1 and x_2 is the number of different cells:

$$d_p^{up}(x_1, x_2) = \sum_{t=2}^{p-1} \sum_{j=2}^{t} [cell_{x_1}(p, t, j) \neq cell_{x_2}(p, t, j)]$$

The upper distance between x_1 and x_2 is defined by the sum of the previous distances up to length n:

$$D_n^{up}(x_1, x_2) = \sum_{p=2}^{n} d_p^{up}(x_1, x_2)$$

Notice that we remove the 2 left columns of cells which use specific transition rules including left border on the left side of the rule.

We compute the upper distance between each pair of new optimal solutions until the length $n = 500$ which allows to clustering the solution according to this distance: two solutions belong to the same cluster if their upper distance is zero. 53 different clusters of solutions with different upper CA computation exist. The size of clusters is not uniform. The size of the five largest ones are respectively 449, 82, 53, 44, and 10, and 37 clusters are composed of a single solution. Fig. 12 shows one of the optimal solutions from the largest cluster. The majority of the optimal solutions found seems to be clustered in one large group which specifies the result on the Hamming distance between optimal solutions (see previous Fig. 8). The relatively small number of clusters allows us to visually analyze the different strategies. Mazoyer's strategy divides recursively the initial problems into sub-problems of size one third of the initial length. Then, each sub-problem is again a synchronization problem of a smaller size. The new optimal solutions also perform a recursive divide-and-conquer strategy that divides the initial line into sub-problem of smaller size. However, the majority of solutions (there is only one exception) divides the initial problem by a factor 2 like in most solutions before Mazoyer's one. Signals with speed of slopes $1/(2^{k-1} - 1)$ with $k \geqslant 1$ are sent instead of the signals of slope $2^{k-1}/(3^k - 2^{k-1})$ with $k \geqslant 1$ of the Mazoyer solution. In the solutions, the signal is then able to divide the initial problem into two sub-problems of same size starting at time step 3/4 of the total number of time steps. Notice that the sub-problems are not identical to the initial problem where one general state is followed by quiescent states for the remaining cells. The two first sub-problems can be more elaborated, but clearly from a visual inspection, the following sub-problems are solved recursively with respect to a scale invariance.

However, although nearly all new solutions shows a new strategy by dividing recursively the initial line into two lines of the same length, one solution shows a Mazoyer-like strategy. Fig. 13 shows an example of space-time diagram of this optimal solution.

6.3. Analysis of the fitness landscape

In this section, we analyze the fitness landscape of the FSSP optimization problem in order to understand why the ILS could find optimal solutions, and possibly to extend the algorithm to other similar optimization problem.

The ruggedness is measured by the fitness cloud (see Section 3.1). Fig. 14 shows the average fitness in the neighborhood of a solution according to the fitness. The sample of solutions is the same as in Section 6.2. It is composed by local optima found along the different runs of ILS. Given that some transition rules are not used in the CA computation, only the neighboring solutions which modify one used transition rule are evaluated, and the average fitness is computed over those neighbors. The Pearson linear coefficient of correlation ρ is high ($\rho = 0.93$), and the R^2 coefficient of the linear regression model is high as well ($R^2 = 0.8697$). The fitness of solutions in the neighborhood is linearly correlated to the fitness of the solution. Even if the

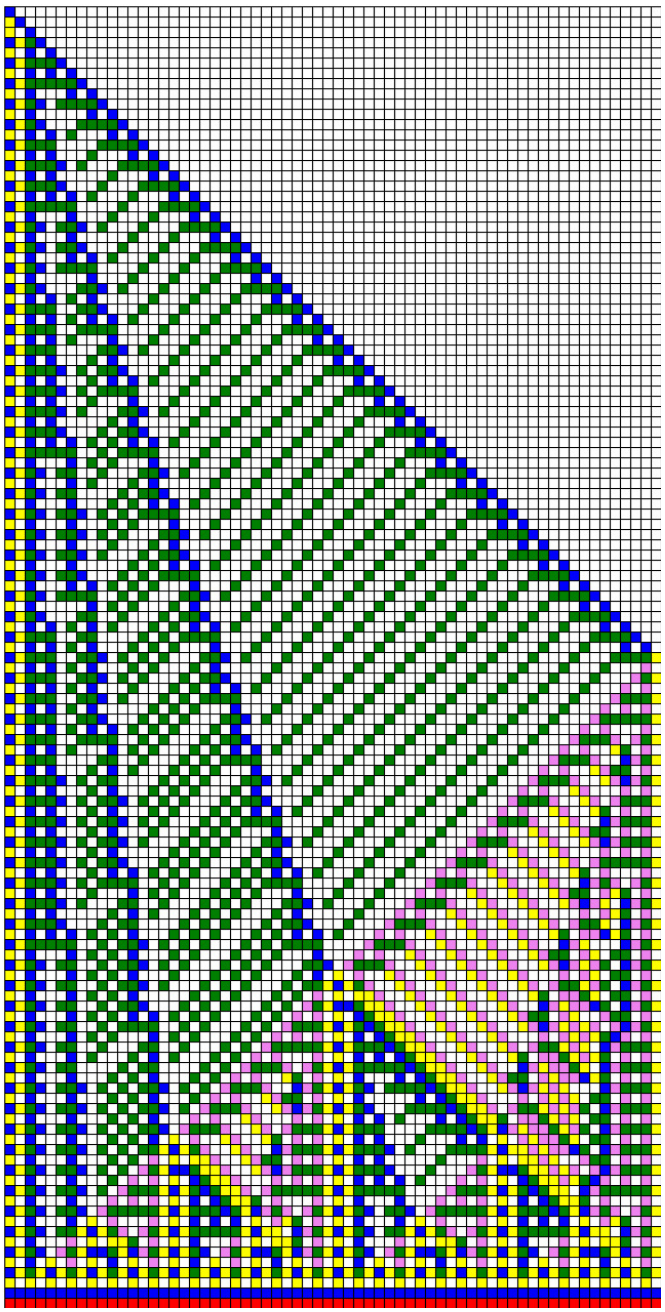

Figure 12: Space-time diagram (cells line of length 64) of an optimal solution from the largest cluster of optimal solutions.

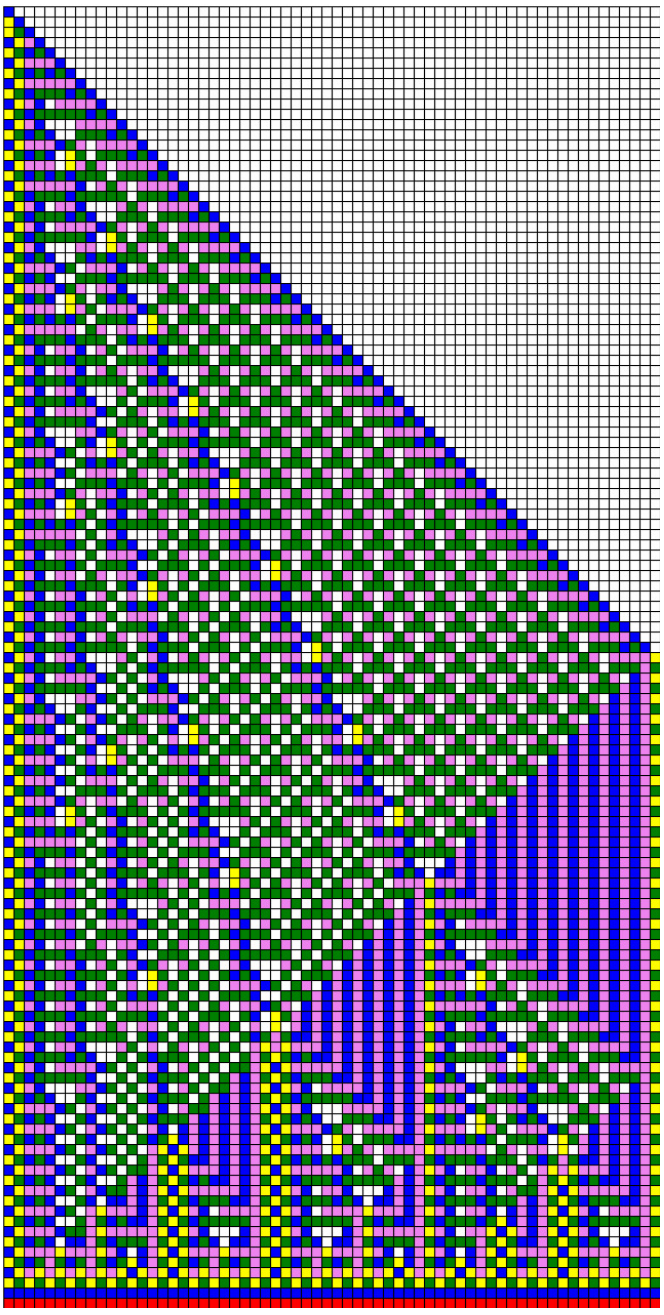

Figure 13: Space-time diagram (cells line of length 64) of the optimal solutions with a Mazoyer-like strategy.

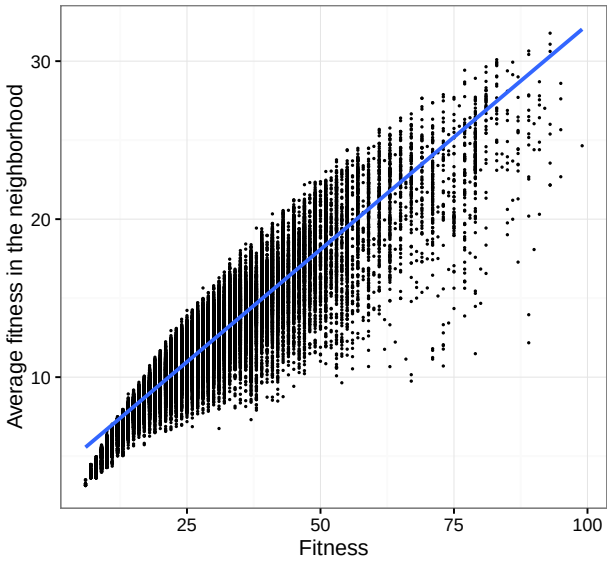

Figure 14: Average fitness in the neighborhood as a function of the solution fitness, and the corresponding regression line.

slope of the regression line (0.284) is lower than the first bisector. This positive high correlation mainly explains the good performance of the local search algorithm: on average, the higher the fitness, the higher the fitness value in the neighborhood. The solutions in the neighborhood are far from random solutions. Indeed, it is easier to find an improving solution in the neighborhood than in the whole search space. Roughly speaking, the structure of the search space of the FSSP problem is not random, and the modification of a few number of transition rules can improve the synchronization of longer cells lines.

In the domain of cellular automata, one knows that when one transition rule of a given transition function is modified, all space-time diagrams can be dramatically changed with no link with the original cellular automaton. Cellular automata are discrete complex systems, and it is nearly impossible to predict the configuration of cells after few steps of computation even with a small number of modifications of the transition function. However, contrary to this classical intuition, the fitness landscape of the FSSP is relatively smooth from the point of view of local search. Although the modification of some transition rules can make the quality drop from high fitness to the minimal fitness value 2, the fitness of neighboring solution is not "random", and it is sufficiently correlated to ensure progressive improvement during the search. The fitness of the solution is a useful information to find a path to an improving solution.

The number of hill-climbings to find improving solution completes the view on the structure of the search and the search process. Fig. 15 shows the average number of hill-climbing procedures to find an improving solution along the ILS on the 200 runs for different values of e_{hc}. The number of evaluations in each hill-climbing e_{hc} has a small impact on the number of HC to improve the current solution. From the fitness 10 to 30, the number of HC increases exponentially with the fitness. It

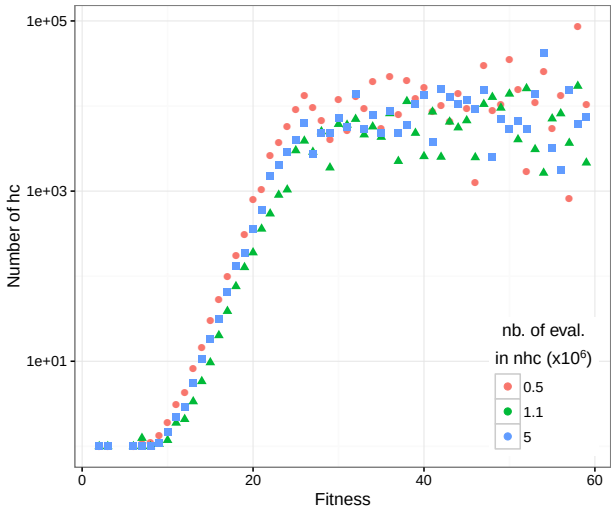

Figure 15: Number of executions of hill-climbing procedure to find an improving solution during the ILS for perturbation strength $k = 3$. Notice the y-log scale.

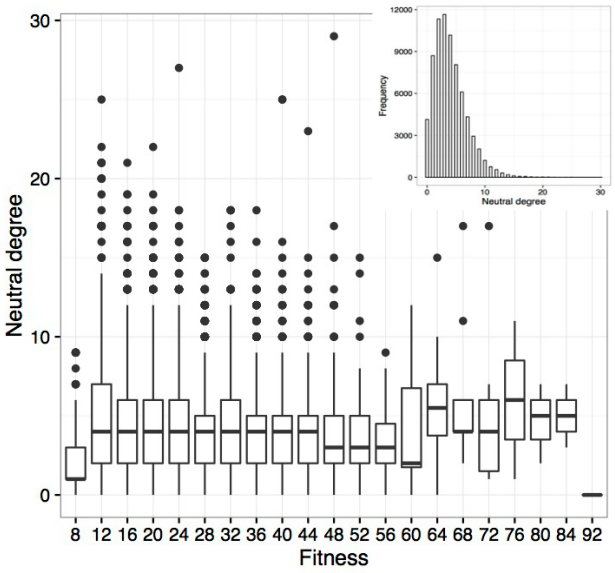

Figure 16: Neutral degree: number of neighbors with the tie fitness value as a function of fitness value. Distribution of neutral degree is given in the upper part of the plot.

becomes exponentially more difficult to find an improving solution. From the fitness 30, the number of hc seems to be constant. This constant value is mainly explained by the maximum number of evaluations of the stopping criterion which stops the algorithm before finding an improving solution. In other words, for high fitness values, when an improving solution is found, the number of *HC* tries have to be small. According to the slope of the fitness cloud, the average fitness in the neighborhood differs linearly from the first bisector $y = x$. It suggests that the probability of finding an improving solution decreases exponentially, and increases exponentially the number of HC tries to find an improvement. Like many other combinatorial problems, the geometrical increase of the number of evaluations should increase the fitness improvement linearly. For the FSSP, the geometrical factor is around 3.34.

The design of the hill-climbing algorithm takes into account the possible tie of fitness values between neighboring solutions. Indeed, the neutrality of the fitness landscape characterized by the presence of plateaus is an important feature of the FSSP optimization problem. Fig. 16 shows the neutral degree, *i.e.* the number of neighboring solutions with the same fitness value (see Section 3.2), as a function of the solution fitness. The sample of solutions is the same as in the Section 6.2 with $72,991$ local optima. Remember that only neighbors modifying one used transition rule are evaluated. The modification of transition rules which are not used in the computation of CA clearly gives a neighbor with a tie fitness value, but it is not counted in the neutral degree of Fig. 16. The neutral degree does not depend on the fitness value of the solution. On average, 4 neighboring solutions have the same fitness value. In comparison to the size of the neighborhood (660 neighbors), the neutral degree rate is small. Nevertheless, few neighboring solutions with the same fitness always give a path to continue the search by using moves on the plateaus of the fitness landscape. This result argues the initial choice of the fitness function which introduces minimal expert information on the FSSP knowledge. The fitness function increases the neutrality with large plateaus, but probably also decreases the local ruggedness of the fitness landscape. The ILS does not stop the search in a strict local optima, but it can continue the exploration of the search space with random moves on plateaus without losing quality.

Another interesting feature of the fitness landscape is the neighborhood of the optimal solutions found. Indeed, it allows us to study the accessibility of the optimal solutions. Fig. 17 shows the density of fitness of solutions in the neighborhood of the optimal solutions. Again, only the neighboring solutions which modify an used transition rule are considered. Optimal solutions are accessible from solutions with the worst fitness: 6.4% of the neighboring solutions has the minimal fitness of 2. The average fitness in the neighborhood is 14.15. The density of fitness is not uniform. Between fitness 2 and 11, the density is high (on average 5%), the density drops after the fitness value 12. The density of solution with odd fitness is higher than solution with even fitness. It suggests that it is more difficult to find solutions that synchronize lines of even length than odd length, but we have no explanation of this observation.

7. Conclusions and perspectives

This paper proposes to solve the 6-states Firing Squad Synchronization Problem (FSSP) using an optimization approach. The FSSP is a classical, still difficult, cellular automata problem where the goal is to find the transition function to synchronize a line of finite automata regardless of its length. Although one hand-made optimal solution designed by Jacques Mazoyer exists, the local search approach used in this paper allows us to find more than seven hundreds of new solutions. Many of them

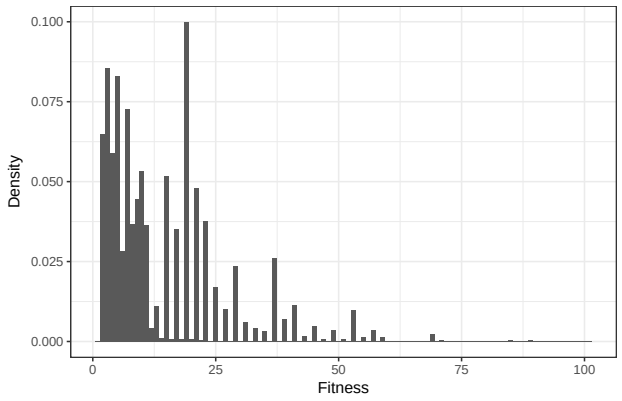

Figure 17: Average fitness density of neighboring solutions of the 718 optimal solutions.

have a lower Kolmogorov complexity and a different computation strategy than the original one.

In order to solve the problem, we proposed a search space and an objective function with minimal information to guide the search towards the goal. We showed that this choice induces a smooth fitness landscape with large plateaus which is relevant for a local search algorithm: better solutions exist in the neighborhood of a solution.

From the features of the fitness landscape, we proposed a dedicated Iterated Local Search algorithm based on a first-improvement hill-climbing with specific acceptance criterion. However, we cannot guarantee that the proposed algorithm is the best choice. Even if the algorithm is able to solve the problem, we argued the relevance of all of its components.

It has been proven that there is no optimal solution for the 4-states FSSP, but it remains an open question to know if there exists an optimal solution for the FSSP with 5 states. This open issue constitutes the main research direction of our methodology which was successful for the 6-states problem. Indeed, it would be possible to use directly the same optimization function, as well as the same optimization algorithm to solve the 5-states variant, or at least to find the transition function which synchronizes the longest line of cells. However, if necessary, the proposed algorithm still has room for improvement. A local search operator could consider the used transition rules differently to the unused transition rules. Clearly, other possible improvements consists of designing a more clever neighborhood operator. Moreover, from the deep analysis of the set of new solutions found in this work, it would be possible to bias the search toward more efficient sub-spaces of solutions.

Finally, our approach is not limited to the 6-states or 5-states FSSP but could be applied to other variants of the Firing Squad Synchronization Problem.

Acknowledgment

The authors would like to warmly thank the anonymous reviewers who greatly helped in improving the former version of the present paper. Enrico Formenti acknowledges partial support by PACA APEX FRI project.

A. Optimal solutions presented in the paper

The following tables give the transition rules of the solutions described in this article. For each solution, each table is made of six sections, one for each of the 6 possible states. Each section is named after the corresponding state and the name can be found in the top left corner. For example, the first section of each table is associated with the state $\bullet$ and if at row r and column l there is a symbol z, it means that $\delta(r, \bullet, l) = z$. When a state combination is unused by the solution, the corresponding entry of the table is left blank. For instance, in the Mazoyer's solution, $\delta(C, \bullet, G) = G$ while the transition $\delta(D, B, B)$ is unused.

The set with the 718 new solutions (with the corresponding id) can be found on the website: `http://www-lisic.univ-littoral.fr/~verel/RESEARCH/firing-squad-synchronization-problem`.

Original solution of Mazoyer (Fig. 2)

•	•	A	B	C	D	$
•	•	•		•	•	•
A	C	B	•	•	•	B
B	A	C	•	•	•	C
C	B	A	•	•	•	A
D	•	•	•	•	•	•
$	•					

A	•	A	B	C	D	$
•			A	A	A	
A	D	F		A	A	F
B	D			A	A	
C	B	B		A	A	B
D	D	A		A	A	A
$	B	F		A	A	

B	•	A	B	C	D	$
•			B	A	•	
A		C		C		C
B	B	D	B	C	D	F
C	B		B			
D	A	C		C	A	C
$			F	A		

C	•	A	B	C	D	$
•	C	A	B	C	A	
A	D	D			D	D
B	D	D			D	D
C	C	D	B	C	D	
D	C	A		C		A
$						

D	•	A	B	C	D	$
•		D	A	•	D	
A	C	A	C	D		A
B	A		D	•	D	
C	•	•	B		•	
D	A	D	B	C	D	
$						

Solution with id 634 with the minimal Kolmogorov complexity (Fig. 9)

•	•	A	B	C	D	$
•	•	C	D	•	•	•
A	A	C	B	•	B	B
B	D		D			
C	C	B	A	C		
D	•				•	
$	•					

A	•	A	B	C	D	$
•	A					
A	•	F				F
B	•	A	D	A	A	D
C	A	A	B	•		
D			B		B	
$	B	F	D			

B	•	A	B	C	D	$
•	•			A		A
A		C		C		C
B						F
C	A			A		
D						
$		C	F			

C	•	A	B	C	D	$
•	•	C	D	C	D	
A		•				
B			B		B	B
C	•	C	D	C	D	
D		B	D		A	A
$	A					

D	•	A	B	C	D	$
•	C	C		C		
A	•	•				
B						
C		C		A		A
D						
$				A		

Solution with id 3 from the largest cluster of optimal solutions (Fig. 12)

•	•	A	B	C	D	$
•	•	C	•	C	D	•
A	A	C	B	•	B	B
B	B	B	A	C	B	A
C	•	C		C	D	
D	D	B	B		C	B
$	•					

A	•	A	B	C	D	$
•	C	D		C	C	
A		F	D	•		F
B	•	B	B		B	B
C	•	•		•	D	
D	C		•	•	D	
$	B	F	B			

B	•	A	B	C	D	$
•	A		C	•	•	A
A		•	•	•		
B	B			B	B	F
C	A	•	C	•	C	•
D	D			•	D	
$	A	B	F			

C	•	A	B	C	D	$
•	•			•		
A	D		A	B		
B	A	A	B	A	B	
C	A	A	D	•	•	
D	•		D	D	D	
$						

D	•	A	B	C	D	$
•	C	A	B	•	B	
A	C	A	•	•		
B	B	•	B	B	B	
C	B				D	
D	•	•		D		
$						

Solution with id 668 using a Mazoyer-like strategy (Fig. 13)

•	•	A	B	C	D	$
•	•	A		C	•	•
A	A	B	•		C	B
B					C	
C				C	C	
D		D	C	C	•	
$	•					

A	•	A	B	C	D	$
•		D			A	
A		F			D	F
B	C		B		B	B
C	D	D		D		
D	C			•	A	
$	B	F	B			

B	•	A	B	C	D	$
•	D			B	A	
A					C	
B						F
C		A		A		A
D		C			C	C
$		B	F	A		

C	•	A	B	C	D	$
•	•	•		D	C	
A	•	D	B	A	B	
B	D	B	A		•	A
C	D	D		•	A	
D	•	B	B	D	C	B
$						

D	•	A	B	C	D	$
•	A	A	D	C	C	
A		D	D	D		
B		A	B	C	A	
C	C	D	D	D		
D	A	A	D		C	
$						

References

[1] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic programming IV: Routine human-competitive machine intelligence*, volume 5. Springer Science & Business Media, 2006.

[2] John R. Koza. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):251–284, 2010.

[3] Bastien Chopard. Cellular automata modeling of physical systems. In *Encyclopedia of Complexity and Systems Science*, pages 865–892. Springer, 2009.

[4] Edward F. Moore. *The Firing Squad Synchronization Problem in Sequential Machines*, volume Sequential Machines. Selected Papers., pages 213–214. Addison-Wesley, Reading MA, 1964.

[5] Jacques Mazoyer. A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50(2):183 – 238, 1987.

[6] Melanie Mitchell, Peter T. Hraber, and James P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993.

[7] Sébastien Verel, Philippe Collard, Marco Tomassini, and Leonardo Vanneschi. Fitness landscape of the cellular automata majority problem: View from the Olympus. *Theoretical Computer Science*, 378(1):54–77, June 2007.

[8] Dietmar Wolz and Pedro Paulo Balbi De Oliveira. Very effective evolutionary techniques for searching cellular automata rule spaces. *Journal of Cellular Automata*, 3(4):289–312, 2008.

[9] Pedro Paulo Balbi de Oliveira. On density determination with cellular automata: Results, constructions and directions. *Journal of Cellular Automata*, 9:357–385, 2014.

[10] Emmanuel Sapin and Larry Bull. Searching for glider guns in cellular automata: Exploring evolutionary and other techniques. In *Artificial Evolution*, pages 255–265. Springer Berlin Heidelberg, 2007.

[11] Ron Breukelaar and Thomas Bäck. Using a genetic algorithm to evolve behavior in multi dimensional cellular automata: Emergence of behavior. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 107–114, New York, NY, USA, 2005. ACM. ISBN 1-59593-010-8.

[12] Fazia Aboud, Nathalie Grangeon, and Sylvie Norre. Improving efficiency of metaheuristics for cellular automaton inverse problem. In *Artificial Evolution*, pages 168–179. Springer, 2013.

[13] M. Bidlo. On routine evolution of complex cellular automata. *IEEE Transactions on Evolutionary Computation*, 20(5):742–754, Oct 2016.

[14] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967. ISBN 0-13-165563-9.

[15] Eiichi Goto. A minimum time solution of the firing squad problem. *Course Notes for Applied Mathematics*, 298, 1962.

[16] Abraham Waksman. An optimum solution to the firing squad synchronization problem. *Information and Control*, 9(1):66 – 78, 1966. ISSN 0019-9958.

[17] Robert Balzer. An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, 10(1):22 – 42, 1967.

[18] Jean-Baptiste Yunes. A propos d'automates cellulaires, suivi par des fonctions booléennes. HDR, 2007.

[19] Hiroshi Umeo, Masashi Maeda, and Kazuaki Hongyo. A design of symmetrical six-state 3n-step firing squad synchronization algorithms and their implementations. In Samira Yacoubi, Bastien Chopard, and Stefania Bandini, editors, *Cellular Automata: 7th International Conference on Cellular Automata, for Research and Industry, ACRI 2006, Perpignan, France, September 20-23, 2006. Proceedings*, pages 157–168, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[20] Peter Sanders. Massively parallel search for transition-tables of polyautomata. In W. Wilhelmi C. Jesshope, V. Jossifov, editor, *Proc. of the VI Int. Workshop on Parallel Processing by Cellular Automata and Arrays*, pages 99–108. Akademie, Berlin, 1994.

[21] Hiroshi Umeo, Jean-Baptiste Yunès, and Naoki Kamikawa. *Cellular Automata: 8th International Conference on Cellular Automata for Reseach and Industry, ACRI 2008, Yokohama, Japan, September 23-26, 2008. Proceedings*, chapter About 4-States Solutions to the Firing Squad Synchronization Problem, pages 108–113. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-79992-4.

[22] Peter F. Stadler. Fitness landscapes. *Biological evolution and statistical physics*, pages 183–204, 2002.

[23] Peter Merz and Bernd Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4):337–352, Nov 2000.

[24] Francisco Chicano, Fabio Daolio, Gabriela Ochoa, Sébastien Verel, Marco Tomassini, and Enrique Alba. Local optima networks, landscape autocorrelation and heuristic search performance. In *Parallel Problem Solving from Nature-PPSN XII*, pages 337–347. Springer, 2012.

[25] Josselin Garnier and Leila Kallel. Efficiency of local search with multiple local optima. *SIAM Journal on Discrete Mathematics*, 15(1):122–141, 2001.

[26] Edward D. Weinberger. Local properties of Kauffman's NK model, a tuneably rugged energy landscape. *Phys. Rev. A*, 44:6399–6413, 1991.

[27] Sébastien Verel, Philippe Collard, and Manuel Clergue. Where are Bottlenecks in NK Fitness Landscapes? In *Evolutionary Computation, 2003. CEC'03*, pages 273–280, Canberra, Australia, December 2003. IEEE Press.

[28] Motoo Kimura. *The Neutral Theory of Molecular Evolution*. Cambridge University Press, Cambridge, UK, 1983.

[29] Erik Van Nimwegen, James P. Crutchfield, and Martijn Huynen. Neutral evolution of mutational robustness. In *Proc. Nat. Acad. Sci. USA 96*, pages 9716–9720, 1999.

[30] Marc Toussaint and Christian Igel. Neutrality: A necessity for self-adaptation. In *Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress - Volume 02*, CEC '02, pages 1354–1359, Washington, DC, USA, 2002. IEEE Computer Society.

[31] Erik Van Nimwegen, James P. Crutchfield, and Martijn Huynen. Metastable evolutionary dynamics: Crossing fitness barriers or escaping via neutral paths? Technical Report 99-07-041, SanteFe institute, 1999.

[32] Mario Garza-Fabre, Eduardo Rodriguez-Tello, and Gregorio Toscano-Pulido. Comparative analysis of different evaluation functions for protein structure prediction under the hp model. *Journal of Computer Science and Technology*, 28(5):868–889, 2013.

[33] Lionel Barnett. Netcrawling - optimal evolutionary search with neutral networks. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 30–37. IEEE Press, 2001.

[34] Sébastien Verel, Philippe Collard, and Manuel Clergue. Scuba Search : when selection meets innovation. In *Evolutionary Computation, 2004. CEC2004*, pages 924 – 931, Portland (Oregon), United States, June 2004. IEEE Press.

[35] Marie-Eleonore Marmion, Clarisse Dhaenens, Laetitia Jourdan, Arnaud Liefooghe, and Sébastien Verel. NILS: a Neutrality-based Iterated Local Search and its application to Flowshop Scheduling. In Peter Merz and Jin-Kao Hao, editors, *11th European Conference on Evolutionary Computation in Combinatorial Optimisation*, volume 6622 of *Lecture Notes in Computer Science*, pages 191–202, Turino, Italy, April 2011. Springer.

[36] El-Ghazali Talbi and Benjamin Weinberg. Breaking the search space symmetry in partitioning problems. *Theoretical Computer Science*, 378(1):78 – 86, 2007.

[37] Eduardo Rodriguez-Tello, Jin-Kao Hao, and Jose Torres-Jimenez. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers and Operations Research*, 35(10):3331 – 3346, 2008.

[38] Fabio Daolio, Marco Tomassini, Sébastien Verel, and Gabriela Ochoa. Communities of Minima in Local Optima Networks of Combinatorial Spaces. *Physica A: Statistical Mechanics and its Applications*, 390(9): 1684 – 1694, July 2011.

[39] Henry A. Kautz, Ashish Sabharwal, and Bart Selman. Incomplete algorithms. *Handbook of Satisfiability*, 185:185–204, 2009.

[40] Matthieu Basseur and Adrien Goëffon. Climbing combinatorial fitness landscapes. *Appl. Soft Comput.*, 30(C):688–704, May 2015.

[41] Gabriela Ochoa, Sébastien Verel, and Marco Tomassini. First-improvement vs. Best-improvement Local Optima Networks of NK Landscapes. In *11th International Conference on Parallel Problem Solving From Nature*, pages 104 – 113, 2010.

[42] John Baxter. Local optima avoidance in depot location. *Journal of the Operational Research Society*, pages 815–819, 1981.

[43] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search: Framework and applications. In *Handbook of Metaheuristics*, pages 363–397. Springer, 2010.